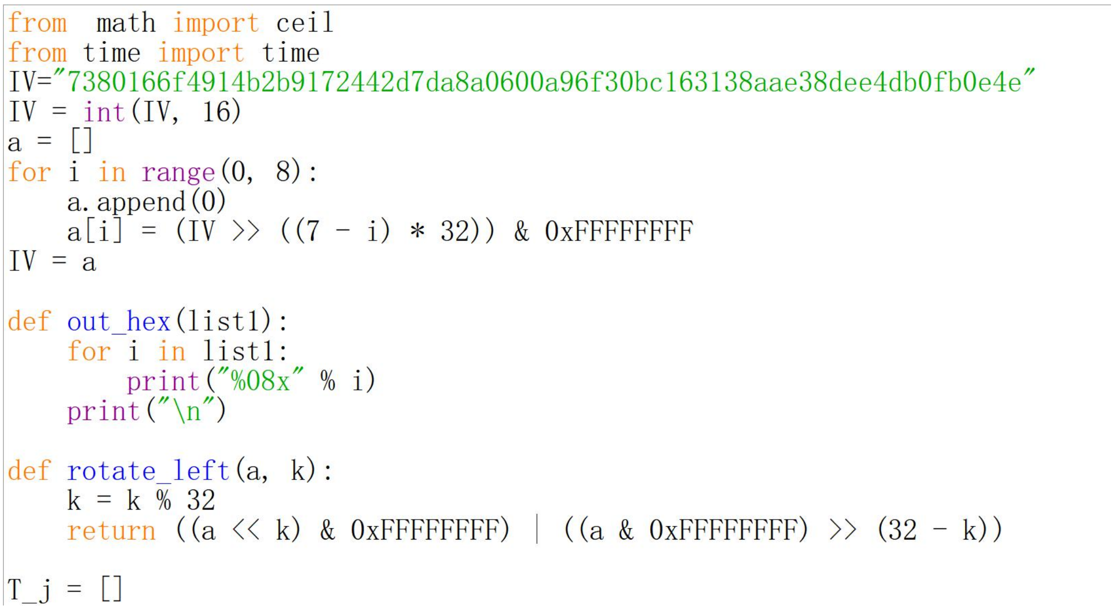
Sm3 生日攻击实验报告

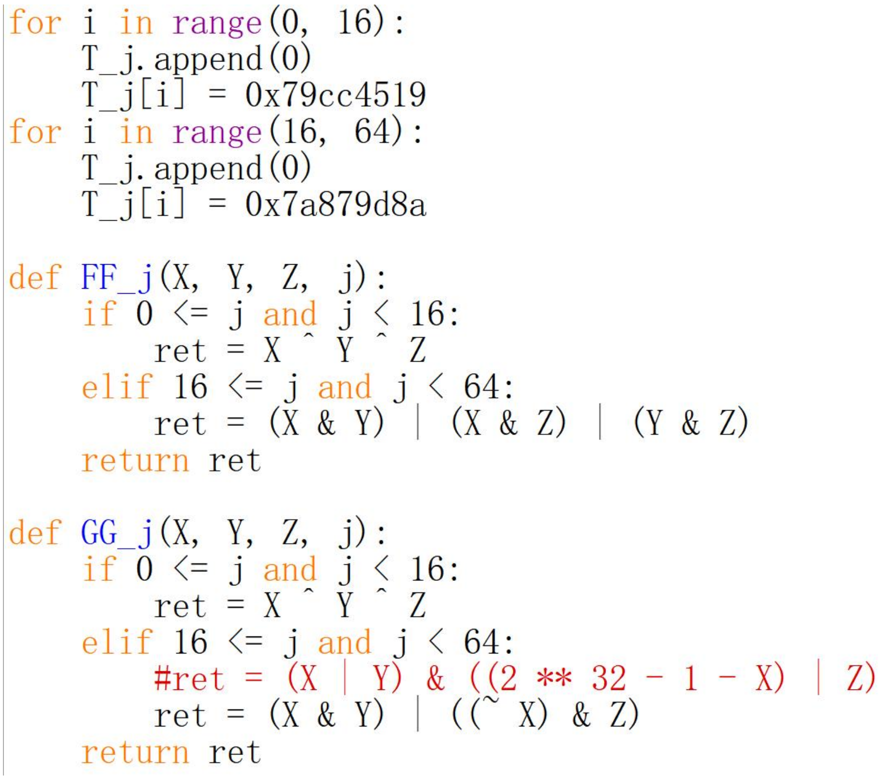
网络空间安全创新创业实践

赵翔正 202000460090

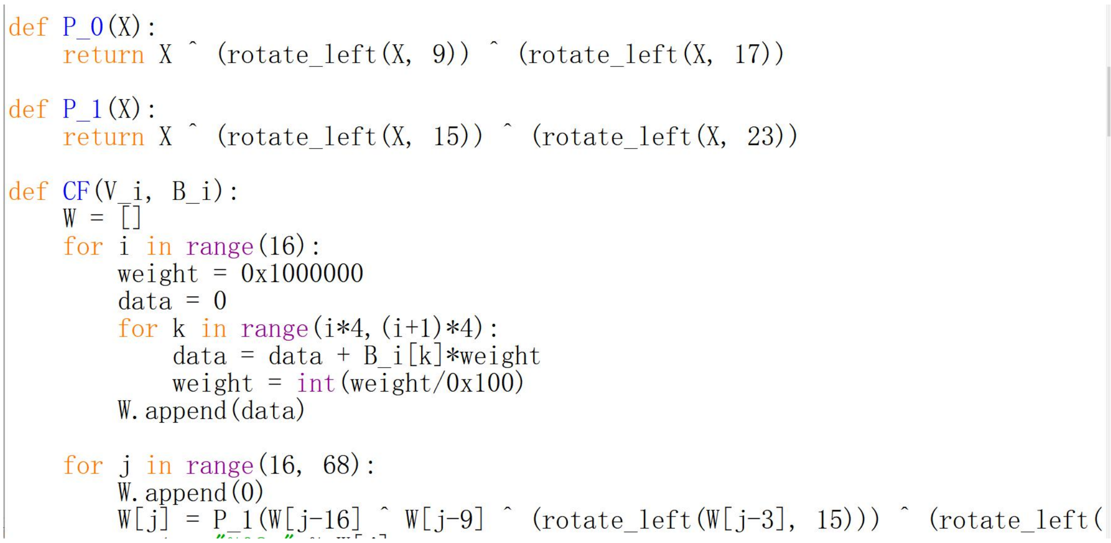
首先实现 sm3 中的各种运算



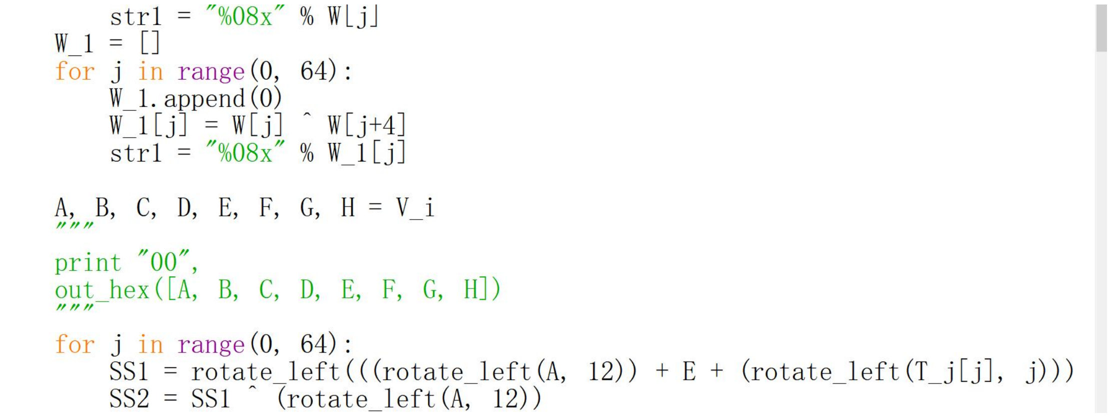
```
from  math import ceil
from time import time
IV="7380166f4914b2b9172442d7da8a0600a96f30bc163138aae38dee4db0fb0e4e"
IV = int(IV, 16)
a = []
for i in range(0, 8):
    a.append(0)
    a[i] = (IV >> ((7 - i) * 32)) & 0xFFFFFFFF
IV = a

def out_hex(list1):
    for i in list1:
        print("%08x" % i)
    print("\n")

def rotate_left(a, k):
    k = k % 32
    return ((a << k) & 0xFFFFFFFF) | ((a & 0xFFFFFFFF) >> (32 - k))

T_j = []
```



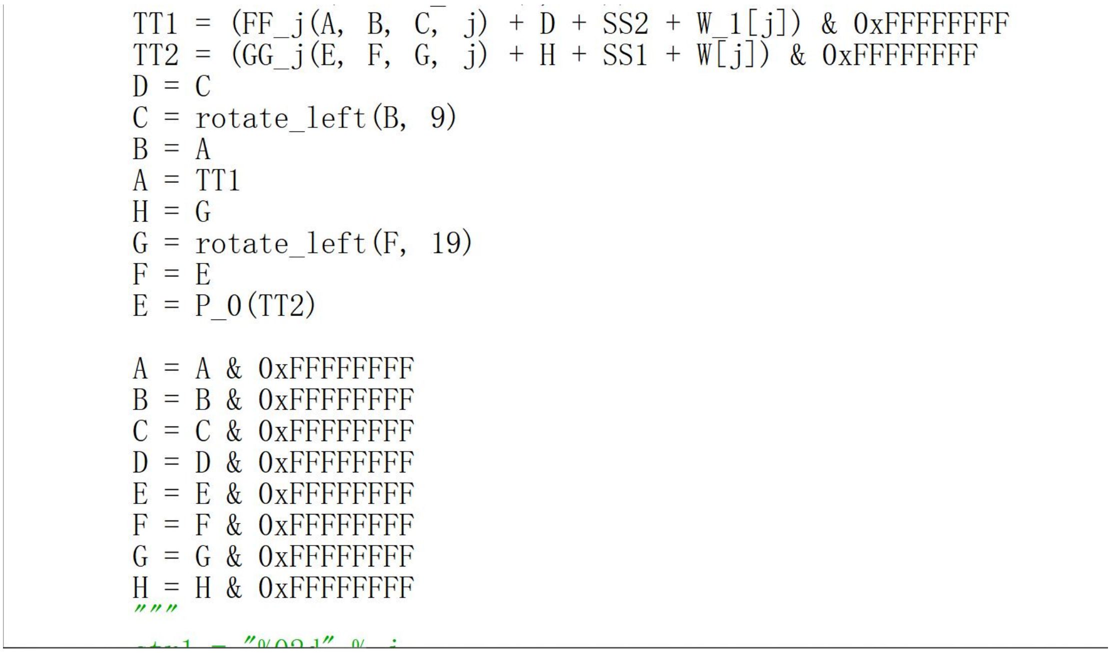
```
for i in range(0, 16):
    T_j.append(0)
    T_j[i] = 0x79cc4519
for i in range(16, 64):
    T_j.append(0)
    T_j[i] = 0x7a879d8a

def FF_j(X, Y, Z, j):
    if 0 <= j and j < 16:
        ret = X ^ Y ^ Z
    elif 16 <= j and j < 64:
        ret = (X & Y) | (X & Z) | (Y & Z)
    return ret

def GG_j(X, Y, Z, j):
    if 0 <= j and j < 16:
        ret = X ^ Y ^ Z
    elif 16 <= j and j < 64:
        #ret = (X | Y) & ((2 ** 32 - 1 - X) | Z)
        ret = (X & Y) | ((~ X) & Z)
    return ret
```



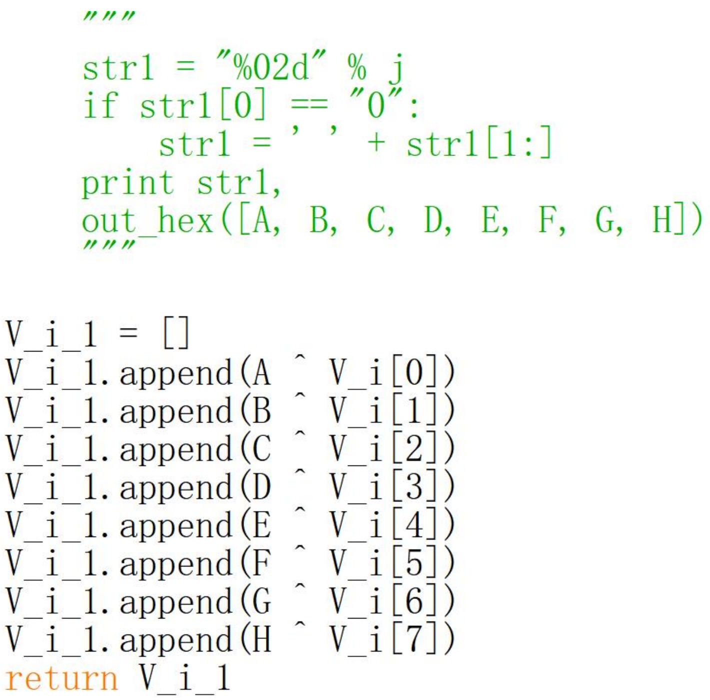
```
def P_0(X):
    return X ^ (rotate_left(X, 9)) ^ (rotate_left(X, 17))

def P_1(X):
    return X ^ (rotate_left(X, 15)) ^ (rotate_left(X, 23))

def CF(V_i, B_i):
    W = []
    for i in range(16):
        weight = 0x1000000
        data = 0
        for k in range(i*4,(i+1)*4):
            data = data + B_i[k]*weight
            weight = int(weight/0x100)
        W.append(data)

    for j in range(16, 68):
        W.append(0)
        W[j] = P_1(W[j-16] ^ W[j-9] ^ (rotate_left(W[j-3], 15))) ^ (rotate_left(
```



```
        str1 = "%08x" % W[j]
    W_1 = []
    for j in range(0, 64):
        W_1.append(0)
        W_1[j] = W[j] ^ W[j+4]
        str1 = "%08x" % W_1[j]

    A, B, C, D, E, F, G, H = V_i
    """
    print "00",
    out_hex([A, B, C, D, E, F, G, H])
    """
    for j in range(0, 64):
        SS1 = rotate_left(((rotate_left(A, 12)) + E + (rotate_left(T_j[j], j)))
        SS2 = SS1 ^ (rotate_left(A, 12))
```

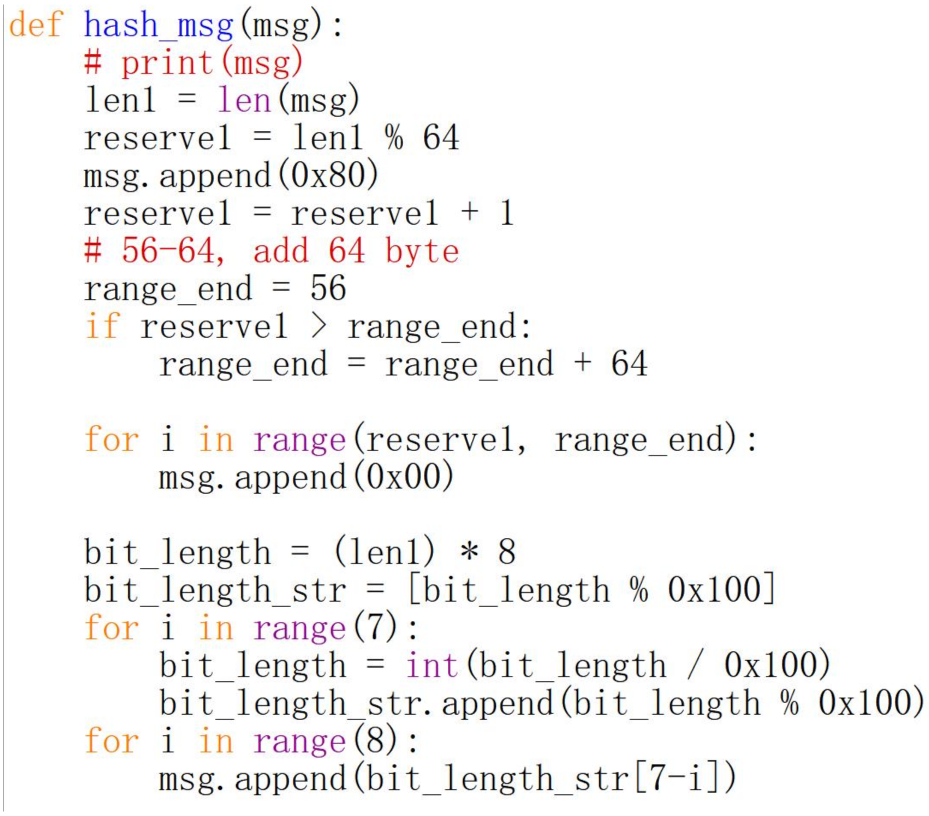


```
        TT1 = (FF_j(A, B, C, j) + D + SS2 + W_1[j]) & 0xFFFFFFFF
        TT2 = (GG_j(E, F, G, j) + H + SS1 + W[j]) & 0xFFFFFFFF
        D = C
        C = rotate_left(B, 9)
        B = A
        A = TT1
        H = G
        G = rotate_left(F, 19)
        F = E
        E = P_0(TT2)

        A = A & 0xFFFFFFFF
        B = B & 0xFFFFFFFF
        C = C & 0xFFFFFFFF
        D = D & 0xFFFFFFFF
        E = E & 0xFFFFFFFF
        F = F & 0xFFFFFFFF
        G = G & 0xFFFFFFFF
        H = H & 0xFFFFFFFF
        """
        str1 = "%02d" % j
```

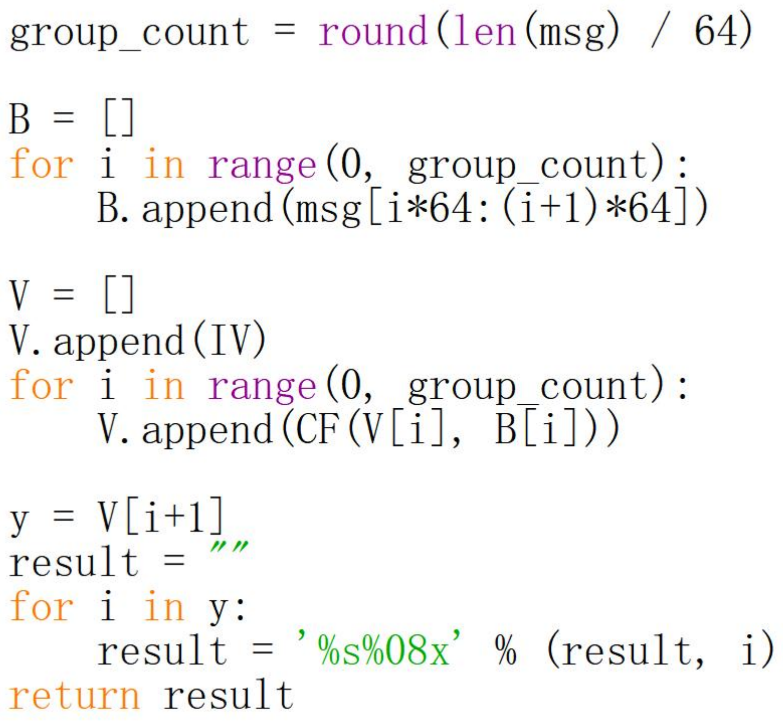


```
        """
        str1 = "%02d" % j
        if str1[0] == "0":
            str1 = ' ' + str1[1:]
        print str1,
        out_hex([A, B, C, D, E, F, G, H])
        """

    V_i_1 = []
    V_i_1.append(A ^ V_i[0])
    V_i_1.append(B ^ V_i[1])
    V_i_1.append(C ^ V_i[2])
    V_i_1.append(D ^ V_i[3])
    V_i_1.append(E ^ V_i[4])
    V_i_1.append(F ^ V_i[5])
    V_i_1.append(G ^ V_i[6])
    V_i_1.append(H ^ V_i[7])
    return V_i_1
```

实现 hash 过程：

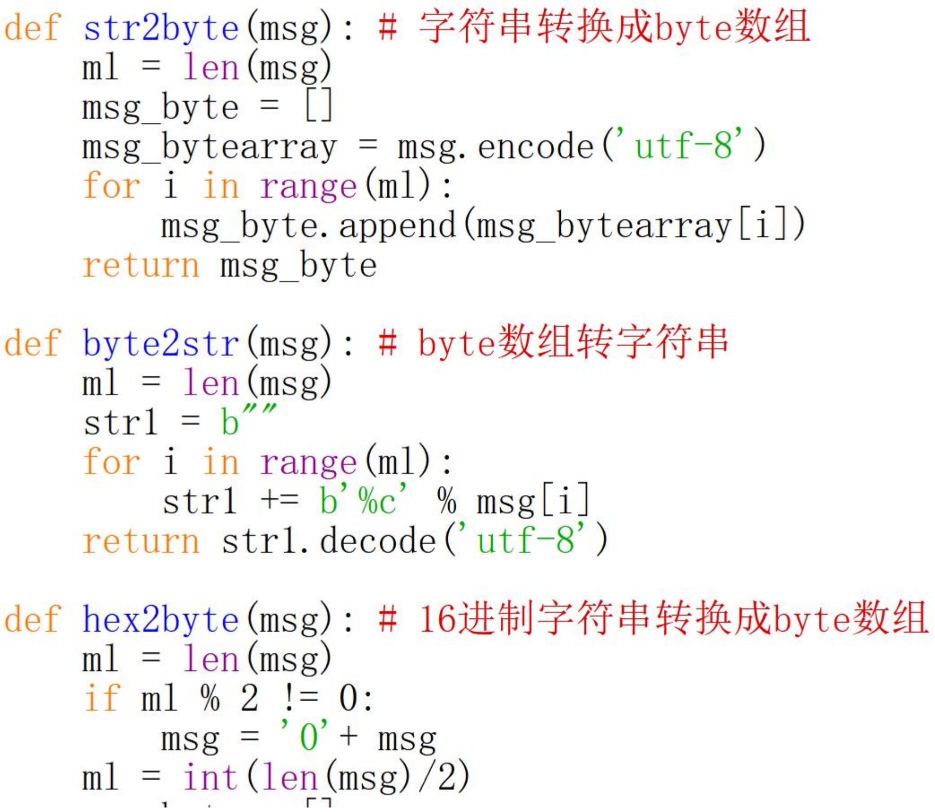


```
def hash_msg(msg):
    # print(msg)
    len1 = len(msg)
    reserve1 = len1 % 64
    msg.append(0x80)
    reserve1 = reserve1 + 1
    # 56-64, add 64 byte
    range_end = 56
    if reserve1 > range_end:
        range_end = range_end + 64

    for i in range(reserve1, range_end):
        msg.append(0x00)

    bit_length = (len1) * 8
    bit_length_str = [bit_length % 0x100]
    for i in range(7):
        bit_length = int(bit_length / 0x100)
        bit_length_str.append(bit_length % 0x100)
    for i in range(8):
        msg.append(bit_length_str[7-i])
```

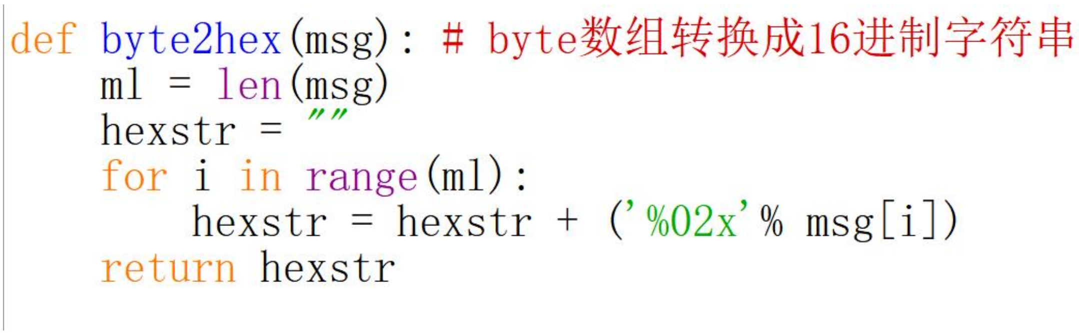


```
group_count = round(len(msg) / 64)

B = []
for i in range(0, group_count):
    B.append(msg[i*64:(i+1)*64])

V = []
V.append(IV)
for i in range(0, group_count):
    V.append(CF(V[i], B[i]))

y = V[i+1]
result = ””
for i in y:
    result = ’%s%08x’ % (result, i)
return result
```

实现字符串、十六进制、字节数组之间的相互转化

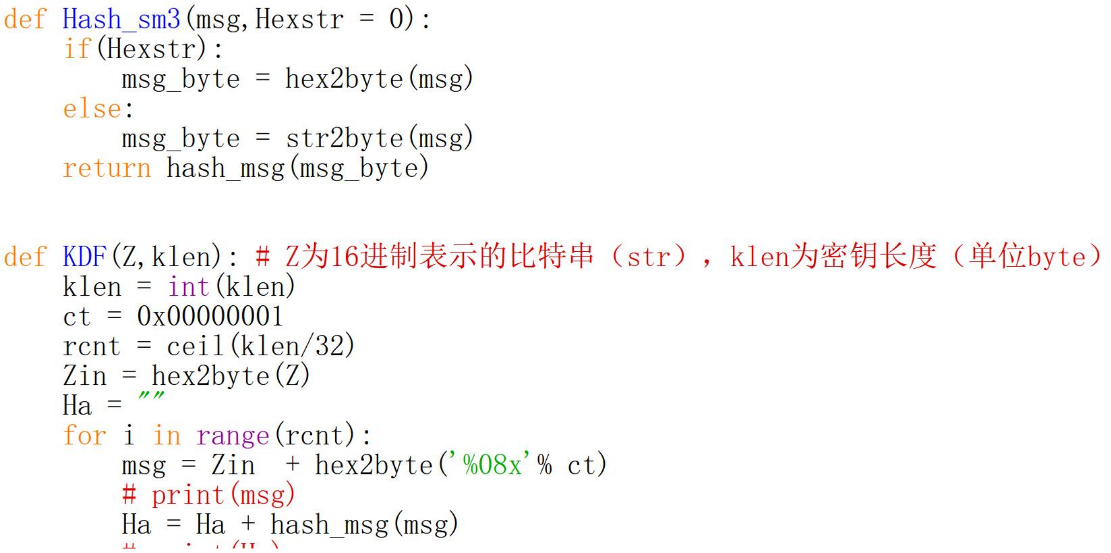


```
def str2byte(msg): # 字符串转换成byte数组
    ml = len(msg)
    msg_byte = []
    msg_bytearray = msg.encode(’utf-8’)
    for i in range(ml):
        msg_byte.append(msg_bytearray[i])
    return msg_byte

def byte2str(msg): # byte数组转字符串
    ml = len(msg)
    str1 = b””
    for i in range(ml):
        str1 += b’%c’ % msg[i]
    return str1.decode(’utf-8’)

def hex2byte(msg): # 16进制字符串转换成byte数组
    ml = len(msg)
    if ml % 2 != 0:
        msg = ’0’+ msg
    ml = int(len(msg)/2)
```



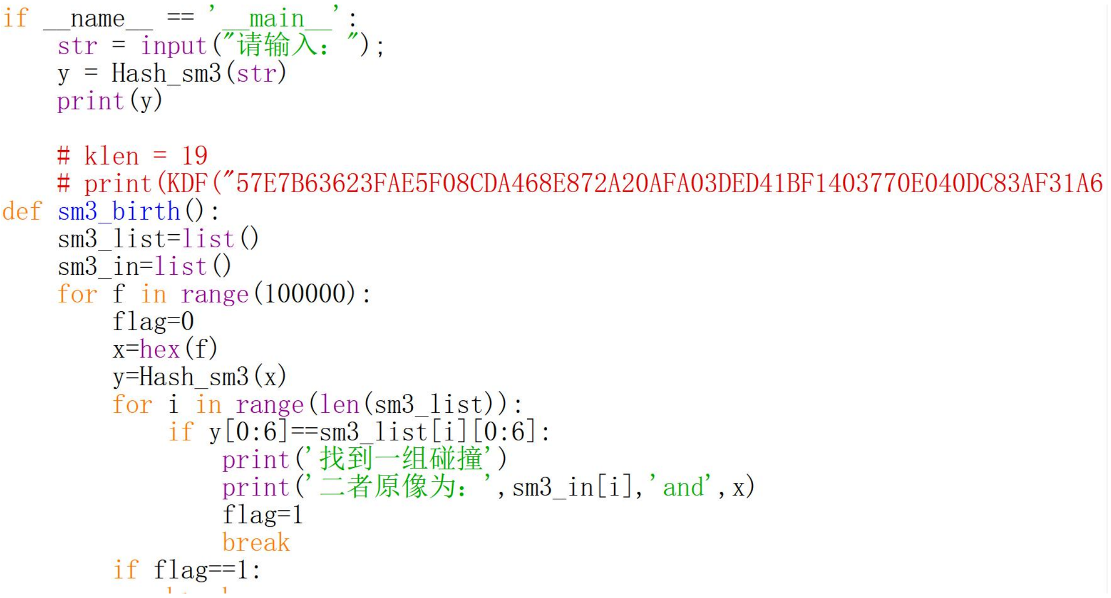
```
def byte2hex(msg): # byte数组转换成16进制字符串
    ml = len(msg)
    hexstr = ””
    for i in range(ml):
        hexstr = hexstr + (’%02x’% msg[i])
    return hexstr
```

实现 sm3 的过程



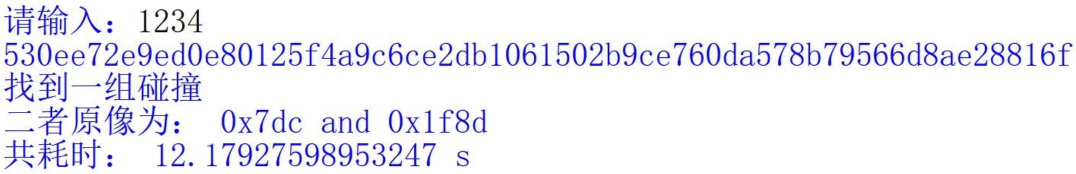
```
def Hash_sm3(msg,Hexstr = 0):
    if(Hexstr):
        msg_byte = hex2byte(msg)
    else:
        msg_byte = str2byte(msg)
    return hash_msg(msg_byte)


def KDF(Z,klen): # Z为16进制表示的比特串（str），klen为密钥长度（单位byte）
    klen = int(klen)
    ct = 0x00000001
    rcnt = ceil(klen/32)
    Zin = hex2byte(Z)
    Ha = ””
    for i in range(rcnt):
        msg = Zin  + hex2byte(’%08x’% ct)
        # print(msg)
        Ha = Ha + hash_msg(msg)
        # print(Ha)
```

执行 sm3 及其生日攻击，由于需要花费的时间过多，此处仅碰撞前 6 位



```
if __name__ == '__main__':
    str = input(”请输入：”);
    y = Hash_sm3(str)
    print(y)

    # klen = 19
    # print(KDF(”57E7B63623FAE5F08CDA468E872A20AFA03DED41BF1403770E040DC83AF31A6
def sm3_birth():
    sm3_list=list()
    sm3_in=list()
    for f in range(100000):
        flag=0
        x=hex(f)
        y=Hash_sm3(x)
        for i in range(len(sm3_list)):
            if y[0:6]==sm3_list[i][0:6]:
                print('找到一组碰撞')
                print('二者原像为：',sm3_in[i],'and',x)
                flag=1
                break
        if flag==1:
            break
            break
        sm3_in.append(x)
        sm3_list.append(y)

t1=time()
sm3_birth()
t2=time()
print('共耗时：',t2-t1,'s')
```

结果如下：



```
请输入：1234
530ee72e9ed0e80125f4a9c6ce2db1061502b9ce760da578b79566d8ae28816f
找到一组碰撞
二者原像为： 0x7dc and 0x1f8d
共耗时： 12.17927598953247 s
```